Universidad Zaragoza

Trabajo Fin de Grado

Título del trabajo: Diseño y despliegue de una arquitectura tolerante a fallos para el aprovisionamiento automático de máquinas físicas en cloud privado

English title: Design and deployment of a fault tolerant architecture to automated provisioning of physical machines for private clouds

Autor

Marta Frías Zapater

Director

Víctor Medel Gracia

FACULTAD DE INFORMÁTICA
2018

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. MARTA FRÍAS ZAPATER,

con nº de DNI 17763102-H. en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado~~/Máster~~) INFORMÁTICA (GRADO), (Título del Trabajo) "DISEÑO Y DESPLIEGUE DE UNA ARQUITECTURA TOLERANTE A FALLOS PARA EL APROVISIONAMIENTO AUTOMÁTICO DE MÁQUINAS FÍSICAS EN CLOUD PRIVADO",

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 18, SEPTIEMBRE, 2018

Fdo: 

Diseño y despliegue de una arquitectura tolerante a fallos para el aprovisionamiento automático de máquinas físicas en cloud privado

RESUMEN

El incremento progresivo del tamaño de los clusters de servicios, así como las necesidades de alta disponibilidad de las aplicaciones o servicios que alojan, obligan a prestar especial atención a requisitos no funcionales, tales como la tolerancia a fallos. Tener un mayor número de servidores trabajando implica un incremento de los potenciales puntos de fallo, por lo que cobra gran importancia la utilización de mecanismos que proporcionen al sistema la capacidad de mantenerse en funcionamiento ante estas situaciones. El objetivo principal es que el servicio pueda seguir operando (aunque sea de un modo degradado, perdiendo prestaciones) aún cuando falle algún componente del sistema.

El proyecto realizado contempla el análisis, diseño y puesta en funcionamiento de un sistema que ofrece el servicio de aprovisionamiento automático de máquinas físicas, garantizando la alta disponibilidad del mismo y centrándose en garantizar la tolerancia a fallos. Durante el desarrollo del trabajo se ha seguido una metodología propia de la ingeniería, verificando cada fase del desarrollo individualmente. Además hay que tener en consideración que se ha trabajado en un entorno en producción, siendo de vital importancia que los pasos a seguir interrumpieran el servicio el menor tiempo posible.

El sistema está formado fundamentalmente por dos nodos que actúan de controladores y de un conjunto de máquinas que son las que se proveen a los clientes. Con el fin de determinar la herramienta más adecuada a las necesidades del entorno de trabajo, se ha realizado un análisis comparativo de diferentes aplicaciones de aprovisionamiento: Ubuntu MAAS, Rack HD y Razor. Posteriormente se ha realizado el diseño de una infraestructura adaptada a las necesidades específicas del caso en concreto para terminar con el despliegue del sistema en alta disponibilidad.

Así mismo, con el objetivo de proporcionar una alta disponibilidad para el servicio se han utilizado dos herramientas adicionales: Pacemaker y Corosync. Aprovechando la funcionalidad de estas aplicaciones se consigue desacoplar la operativa de la base de datos del servicio completo permitiendo a su vez definir el uso de una dirección IP virtual que se intercambia entre los servidores en función de si están actuando como máquina principal o secundaria. Finalmente se han planteado diferentes escenarios con el objetivo de realizar una evaluación completa del sistema y determinar la correcta funcionalidad del mismo.

Palabras clave: alta disponibilidad, cloud privado, aprovisionamiento automático, failover.

Tabla de contenidos

1. Introducción

El auge del paradigma de computación cloud en los últimos años, y la problemática de gestión y procesado de datos asociada al Big Data, hace que determinados requisitos no funcionales de aplicaciones y servicios cobren mayor importancia. En este contexto, la tolerancia a fallos resulta una característica fundamental en dichos sistemas, prestando especial atención al concepto de replicación [1]. Cuando se gestionan recursos críticos es un aspecto fundamental asegurar su fiabilidad y que permanecen disponibles el mayor tiempo posible. Para lograr dicho objetivo, se hace uso de la redundancia, de manera que si un componente falla otro puede reemplazarlo. En las grandes empresas o corporaciones tiene tal trascendencia este hecho que incluso se aseguran que la replicación se distribuya a nivel geográfico. De esta forma se garantiza que si ocurre un desastre en una zona concreta (por ejemplo un terremoto o un incendio) los recursos van a seguir estando accesibles [2]. En cualquier negocio de estas características, una interrupción en sus sistemas puede suponer grandes pérdidas, a nivel productivo por consiguiente a nivel de ingresos. Con esta motivación se focalizan en implementar estrategias e infraestructuras para lograr servicios de alta disponibilidad. Es necesario tener mecanismos que proporcionen la capacidad de monitorizar el sistema para detectar los fallos de la manera más rápida posible, permitiendo realizar una recuperación y conseguir que el sistema siga en funcionamiento.

La redundancia junto con el concepto de alta disponibilidad permiten asegurar cierto nivel de tolerancia a fallos y por ende, continuidad operativa. No obstante obtener estas capacidades para mejorar la disponibilidad de un servicio, conseguir equilibrio de carga y redundancia en los servicios supone todo un desafío [3].

Una gran parte de las empresas actuales se benefician de las ventajas de elasticidad que proporcionan los servicios en la nube. Esta tendencia tecnológica se conoce como cloud computing, y permite reducir los costes, la complejidad de configuración y administración de los niveles más bajos de la infraestructura. Sin embargo, y desde el punto de vista de los proveedores, el coste puede no ser evidente a priori pero conseguir un sistema altamente fiable y tolerante a fallos supone realizar grandes inversiones.

La nube se puede entender como una capa conceptual que permite a los clientes visualizar los recursos disponibles, tanto software como hardware, de una forma transparente [3]. Los modelos de servicios en la nube se dividen básicamente en tres: IaaS (infraestructura como servicio), PaaS (plataforma como servicio), SaaS (software como servicio). Se basan en ofrecer servidores de alojamiento compartido a los clientes finales. Sin embargo, existe un elemento a más bajo nivel que permite disponer de hardware dedicado y que es denominado hardware como servicio o metal como servicio (MaaS). Este modelo es utilizado especialmente en el cloud privado dado que permite aumentar la seguridad (ya que el servidor no se comparte por diferentes clientes) además de ofrecer posibilidades ilimitadas de administración (cosa que es imposible en los otros modelos de servidores en la nube).

El objetivo principal de este proyecto es implementar un cloud privado en alta disponibilidad garantizando la tolerancia a fallos. Es imprescindible señalar que las soluciones que se pueden encontrar en el caso de un cloud público no son extrapolables en el caso de un cloud privado. Hay múltiples opciones posibles pero la selección de la solución más apropiada no es algo trivial y hay que tener en consideración las implicaciones que conlleva a todos los niveles. Por ello y tras un exhaustivo análisis se han considerado diferentes herramientas y soluciones adaptándolas a las necesidades específicas del proyecto. Más concretamente, se ha seleccionado en primer lugar una tecnología que permitiera realizar un aprovisionamiento automático de máquinas físicas. Dicha aplicación almacena su información en una base de datos, por lo que para asegurar la tolerancia a fallos y la continuidad operativa es necesario añadir una herramienta adicional que se ocupe de proporcionar la capacidad de ofrecer dicho servicio en alta disponibilidad.

Cabe mencionar que para desarrollarlo ha sido necesario apoyarse en los conceptos de administración de sistemas analizando las consecuencias que suponía cada configuración, tanto a nivel de máquinas físicas y los servicios alojados en las mismas como a nivel de red.

La motivación de este proyecto se ha fundamentado principalmente en el interés actual de cada vez más empresas de utilizar y gestionar sistemas en alta disponibilidad. Hay que destacar el hecho de que se ha trabajado totalmente en un clúster en fase de producción, que ha estado continuamente utilizado por clientes (usuarios). Esto significa que se ha tenido que seguir una metodología concreta, asegurando el continuo funcionamiento del sistema e interrumpiendo el servicio el menor tiempo posible. Este es un aspecto se convierte en crucial cuando se realiza en un ámbito profesional, dentro de una empresa.

La memoria del proyecto se estructura en 7 capítulos, empezando por la presente introducción. En los Capítulos 2 y 3 se analiza el marco conceptual y el background tecnológico respectivamente. A partir del análisis de ambos se realiza el diseño del sistema en el Capítulo 4 y su despliegue se detalla en el Capítulo 5. A continuación se evalúa dicho sistema en el Capítulo 6. Finalmente, el Capítulo 7 recoge las conclusiones más relevantes.

2. Marco conceptual

2.1 Clúster de alta disponibilidad (HA)

El término **clúster** hace referencia a un conjunto de sistemas informáticos que trabajan conjuntamente con un objetivo común, de forma transparente. Se pueden catalogar en cuanto a las características características que ofrecen: alto rendimiento, balanceo de carga o escalabilidad [4]. En el caso que se trata en este proyecto se persigue el objetivo de obtener una alta disponibilidad que garantice la tolerancia a fallos de un servicio concreto.

La idea de redundancia[1], junto con la de alta disponibilidad[2], establecen la capacidad de un sistema para detectar un fallo de la forma más rápida posible y que tenga la aptitud de recuperarse afectando lo menos posible al servicio.

Más específicamente, un clúster de alta disponibilidad es un sistema orientado a ofrecer y garantizar servicios en alta disponibilidad, es decir, con **alto grado de fiabilidad** y de **continuidad operativa**. Se basa en un conjunto de máquinas interconectadas y monitorizadas entre sí, que tienen la capacidad de asumir el servicio cuando algún componente del sistema falla, asegurando el correcto funcionamiento. Cuando se habla de "fallo", se puede hacer referencia tanto a una máquina que ha dejado de funcionar como a un cable desconectado u otras circunstancias que comprometan la red de comunicaciones. Debe ser capaz de detectar estas situaciones y mantener el servicio **sin intervención humana**, garantizando la integridad de los datos del clúster.

El tamaño mínimo para obtener un clúster de alta disponibilidad es de dos nodos (mínimo para obtener redundancia).

Por otra parte, para cualquier negocio, una interrupción de sus sistemas puede suponer un grave problema, ya que la principal consecuencia es la pérdida de productividad y con ello, pérdidas de ingresos. Como es obvio se desea que los sistemas sean fiables y permanezcan disponibles la mayor parte del tiempo posible. Por ello y centrando la atención en la alta disponibilidad, el beneficio principal es la **tolerancia a errores** que se puedan producir en los nodos que forman el clúster. Si una máquina falla, otra se encarga de seguir ofreciendo el servicio. Sin embargo, **es imposible garantizar una disponibilidad del 100% en una infraestructura**, incluso si ésta es de alta disponibilidad. Por este motivo debe estar preparada para poder recuperarse en el menor tiempo posible y con la menor pérdida de datos. Es de vital importancia contar con un protocolo que garantice la **capacidad de recuperación**.

[1] Redundancia: replicación de aquel software o hardware para el que se desea asegurar una tolerancia ante los posibles fallos.
[2] Alta disponibilidad: protocolo de diseño e implementación de un sistema que asegura cierto nivel de continuidad operativa durante un tanto por ciento de un tiempo determinado.

Finalmente, pueden considerarse otras ventajas relacionadas con poseer un clúster de alta disponibilidad como podría ser reducir costes frente a equipos con tolerancia a fallos en todos los componentes (redundancia hardware), reducir tiempos de inactividad planificados al poder apagar un equipo temporalmente sin dejar de dar servicio o la posibilidad de recuperación ante fallos.

2.1.1 Tipos de clusters HA

Se pueden diferenciar fundamentalmente dos tipos de clusters que ofrecen alta disponibilidad en base a los fallos que pueden detectar.

- **Alta disponibilidad de infraestructura**: El software de alta disponibilidad permite poner en funcionamiento un nodo si se produce un fallo a nivel de hardware en alguna de las máquinas del clúster con el objetivo de que el sistema siga funcionando correctamente. Los servicios deben migrarse a la máquina original cuando se recupera. Garantiza la disponibilidad del servicio.

- **Alta disponibilidad de aplicación**: Detecta si se produce un fallo de hardware o a nivel de aplicación en alguno de los nodos, y es capaz de poner en funcionamiento los servicios que han fallado en otra de las máquinas del clúster. Garantiza además la integridad de los datos.

Las **configuraciones** más comunes para los clusters de alta disponibilidad son la de activo/activo y activo/pasivo.

- **Activo/activo**: Los recursos son ejecutados de forma simultánea en todos los servidores disponibles, de forma que los clientes pueden acceder a cualquiera de las máquinas. Si un nodo del clúster falla, los servicios siguen estando accesibles (*Fig.1*).

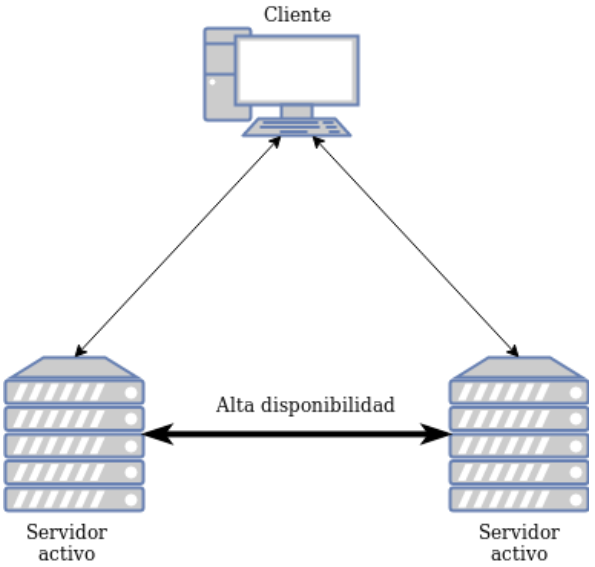

Fig. 1. Clúster activo-activo.

- **Activo/pasivo**: Los recursos únicamente están accesibles a través de uno de los servidores. El resto de nodos pueden acceder a la información del principal, sin embargo no se activan mientras el principal esté disponible (*Fig.2*).

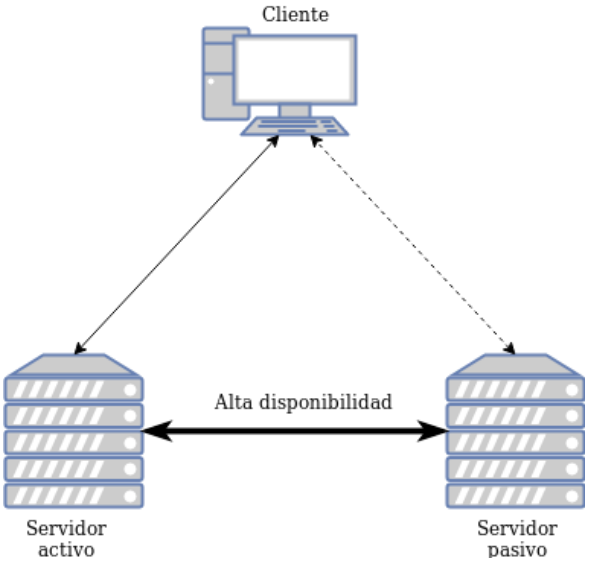

Fig. 2. Clúster activo-pasivo.

3. Background tecnológico

El *cloud computing*[3] han cobrado mucha importancia en los últimos años, y cada vez es mayor el número de empresas y organizaciones que deciden ubicar sus sistemas en la nube. Esto les permite reducir las inversiones para mantener las aplicaciones, plataformas o servidores propios; ofrece flexibilidad, escalabilidad y facilidad de configuración además de reducir la complejidad permitiendo abstraerse de los niveles más bajos de la infraestructura tecnológica. El modelo más conocido es el **cloud público**, en el que se ofrecen servidores dentro de un entorno virtualizado a través de internet y los usuarios se benefician de una infraestructura compartida. Los modelos de servicio en la nube son fundamentalmente tres: infraestructura como servicio (**IaaS**), plataforma como servicio (**PaaS**) y software como servicio (**SaaS**). Sin embargo, existe otro elemento a más bajo nivel que es el servidor **Bare Metal** y que se puede definir como metal como servicio (**MaaS**) o hardware como servicio (**HaaS**). Permite realizar el aprovisionamiento de forma dinámica y flexible de grupos de recursos hardware. La diferencia es que son máquinas cuyos recursos están disponibles únicamente para un cliente en vez de los clásicos servidores de alojamiento compartido [5]. Este modelo es el que se establece en el **cloud privado** (aunque a su vez puede ofrecer los modelos de servicios de IaaS y PaaS). La ventaja principal son las posibilidades de administración ilimitadas y el acceso a hardware dedicado, además de ofrecer un nivel más alto de seguridad.

En estos entornos se acostumbra a trabajar con un elevado número de máquinas, que dependiendo de los ámbitos pueden incluso requerir cientos o miles de servidores. Hay que tener en cuenta que en estos casos no resulta práctico trabajar físicamente “in situ” en cada sistema. Por ello y para facilitar el trabajo de los administradores de estos sistemas existen herramientas que permiten agilizar la instalación y configuración de los mismos, permitiendo gestionar infraestructuras de cualquier tamaño. Por otra parte, cuando se habla de *Cloud* es drásticamente necesario disponer de un mecanismo que permita realizar un aprovisionamiento automático de máquinas físicas y hay que valorar las posibles alternativas [7].

En el apartado siguiente se presentan los requisitos concretos que se han establecido, con el fin de seleccionar la herramienta que más se aproxime a las necesidades de este proyecto.

3.1 Requisitos

El objetivo de este apartado se centra en analizar las herramientas existentes que posibiliten realizar un aprovisionamiento automático de máquinas físicas cumpliendo los siguientes requerimientos:

[3] Cloud computing: Es un modelo que permite el acceso a la red ubicuo, conveniente y bajo demanda a un grupo compartido de recursos informáticos configurables (por ejemplo, redes, servidores, almacenamiento, aplicaciones y servicios) que pueden aprovisionarse y liberarse rápidamente con un mínimo esfuerzo administrativo o interacción del proveedor de servicios [6].

- Puesta en marcha de una máquina desde cero, instalando el sistema operativo y dejándola operativa.
- Encendido/apagado de máquinas de forma remota y bajo demanda.
- Gestión de red de manera automática y a varios niveles (gestión de VLAN).
- Gestión de máquinas físicas y virtuales de forma transparente.
- Oferta de servicios adicionales como DNS[4] y DHCP[5].
- Posibilidad de configuración en alta disponibilidad.

3.2 Tecnologías de Metal as a Service

Existen múltiples tecnologías que permiten realizar un aprovisionamiento de máquinas. Esto consiste básicamente en ser capaz de preparar un servidor con el software y los datos necesarios para que se puedan ejecutar servicios. Dichas herramientas ayudan a automatizar la infraestructura creando y administrando sus entornos, y asegurando configuraciones consistentes. El estudio se va a centrar en aquellas aplicaciones disponibles que permitan realizar un aprovisionamiento automático y cumplan los requisitos que se han planteado anteriormente.

Las opciones existentes son numerosas, por lo que a continuación se presentan tres: **Ubuntu MAAS**, **Rack HD** y **Razor**.

3.2.1 Ubuntu MAAS

MAAS significa *Metal As A Service* y permite trabajar con servidores físicos de la misma manera que si se tratara de máquinas virtuales, convirtiéndolos en un recurso elástico similar a una nube. Esta herramienta permite descubrir automáticamente recursos hardware conectados a la red, así como aprovisionar y destruir las máquinas rápidamente. Controla las máquinas mediante el protocolo IPMI[6] (u otro BMC[7]) y se basa en el mecanismo de PXE[8] para incorporar los recursos al sistema junto con imágenes predefinidas para instalar en una máquina desde cero. Dispone de unos test que permiten conocer el estado del hardware e informa de si se producen fallos (por ejemplo a nivel de disco duro). Ofrece los servicios de DHCP y DNS a los dispositivos conectados a la red, además del servicio de NTP[9] a la

[4] Domain Name System (DNS): Sistema de resolución de nombres de dominio..
[5] Dynamic Host Configuration Protocol (DHCP): Protocolo de configuración dinámica de direcciones IP y otros parámetros de configuración de red.
[6] Intelligent Platform Management Interface (IPMI): Conjunto de especificaciones que definen las interfaces utilizadas para administrar hardware de forma remota.
[7] Baseboard Management Controller (BMC): Microcontrolador integrado en la placa que proporciona la inteligencia en la arquitectura IPMI. Gestiona la interfaz entre el software de administración del sistema y el hardware de la plataforma.
[8] Preboot eXecution Environment (PXE): Entorno para arrancar e instalar un sistema operativo a través de una red, de manera independiente de los dispositivos de almacenamiento o los sistemas instalados.
[9]Network Time Protocol (NTP): Protocolo de sincronización de los relojes de los sistemas informáticos a través de la red.

infraestructura que controla. Por otra parte, además de una interfaz de línea de comandos, proporciona una interfaz gráfica de usuario y una API[10] (que permite vincular a MAAS en la arquitectura de automatización). Finalmente, y como punto clave para éste proyecto, permite obtener una configuración de alta disponibilidad de forma sencilla ya que posibilita que las aplicaciones de MAAS instaladas en diferentes máquinas se integren de una forma semi-automática.

3.2.2 Razor

Es una aplicación de código abierto que permite la gestión y el aprovisionamiento de máquinas físicas y virtuales, centrada en políticas. Esto quiere decir que previamente el administrador debe especificar una serie de reglas que se deben cumplir y aplicar a los nodos una vez se vayan a agregar al sistema. Esta particularidad permite usar las características del hardware para tomar decisiones de aprovisionamiento. Utiliza PXE junto con una imagen especial del microkernel de Razor para poner en marcha las máquinas. Está diseñado para funcionar integrándose con Puppet[11], pero se recomienda que se instalen en servidores separados para evitar conflictos con los puertos. Sin embargo, hay que tener en cuenta que el soporte para la utilización de otras herramientas está limitado a las soluciones diseñadas por los usuarios. Por otra parte, si se quiere obtener una alta disponibilidad es necesario que el administrador diseñe el sistema por su cuenta, replicando la base de datos, replicando los servidores, etc. de forma manual ya que no ofrece ninguna integración entre aplicaciones de Razor.

3.2.3 Rack HD

Es una herramienta de código abierto que automatiza el descubrimiento, la descripción, el aprovisionamiento y la programación de hardware (servidores, switches y almacenamiento) independientemente de la plataforma. El objetivo principal de Rack HD es proporcionar APIs REST[12] para administrar recursos hardware de forma automatizada. Se comunica directamente con la placa sobre el protocolo IPMI y utiliza PXE para la puesta en marcha de los servidores. Se basa en combinar herramientas de código abierto con un motor de flujo de trabajo declarativo basado en eventos, similar a Razor. La característica principal es que proporciona la posibilidad de desarrollar flujos de trabajo personalizados y utilizar la API REST para pasar detalles de configuración de forma dinámica. En cuanto a servicios adicionales que ofrece, tiene la capacidad de adaptarse a diferentes entornos de red para los servicios TFTP[13] y DHCP, tanto integrados en el servidor de Rack HD como en un servidor separado. Por otra parte ofrece la posibilidad de configurar el servicio UCS[14] que permite comunicarse con dispositivos Cisco. En cuanto a la alta disponibilidad, igual que en

[10] Application Programming Interface (API): Conjunto de definiciones, protocolos y funciones o métodos que proporciona una aplicación para ser utilizada por otro software.
[11] Puppet: Herramienta de código abierto que permite gestionar la configuración de equipos informáticos que permite automatizar tareas y monitorizar estados finales
[12] API REST: Arquitectura software utilizada para diseñar aplicaciones web.
[13] Trivial File Transfer Protocol (TFTP): Protocolo simple de transferencia de archivos.
[14] Unified Computer System (UCS): Productos informáticos Cisco para servidores. El servicio UCS de Rack HD ofrece un protocolo de comunicación orientado a estos dispositivos.

el caso de Razor, es el administrador el que debe buscar una solución que proporcione dicha funcionalidad.

En la *Fig.3* se muestra una tabla comparativa con los puntos claves de las tres alternativas que se han valorado en este apartado.

	MAAS	Rack HD	Razor
Gestión de recursos	Encendido/apagado, instalación de SO, configuración básica de recursos	Encendido/apagado, instalación de SO, actualización de firmware y BIOS, configuración básica de recursos	Encendido/apagado, instalación de SO y software, configuración básica de recursos
Gestión de red	VLAN[15] y fabric	No	No
Configuración de alta disponibilidad	Integrada en la aplicación	El administrador debe buscar la forma de hacerlo	El administrador debe buscar la forma de hacerlo
Servicios adicionales	DNS, NTP, DHCP	TFTP, DHCP, UCS	No
Despliegue	Máquinas físicas, máquinas virtuales, contenedores	Máquinas físicas	Máquinas físicas, máquinas virtuales
Detección de fallos	Test hardware (computación y almacenamiento)	Test hardware (computación y almacenamiento)	Mediante flujos de trabajo
Interacción	GUI / API / CLI[16]	GUI / API	API / CLI
Compatibilidad	Windows, Ubuntu, CentOS, RHEL[17]	Independiente de la plataforma	Servidor: RHEL y CentOS Cliente: Windows, RHEL, CentOS, Ubuntu y Debian

Fig. 3. Tabla comparativa de las plataformas de aprovisionamiento.

Existen otras alternativas o sistemas integrados que no se han analizado y de las que pueden encontrarse referencias y comparativas. Además en este caso no se ha realizado una comparación a nivel de rendimiento [8].

3.3 Selección de plataforma de aprovisionamiento

Llegados a este punto cabría plantearse la siguiente pregunta: **¿Éstas herramientas son un paso adelante respecto al software como Chef, Puppet o Ansible?**

Depende de las necesidades de cada sistema pero ninguna de las anteriores se podría considerar como un sustituto ya que trabajan a más bajo nivel. Hay que plantearse las ventajas y desventajas de cada aplicación, en base a los requerimientos del entorno que se desea gestionar. En primer lugar, cabe destacar que con éstas herramientas no es necesario programar en lenguajes como Ruby o Python. Por otra parte, hay que tener en cuenta que tanto Chef como Puppet y Ansible son herramientas de gestión de la configuración muy personalizables y se obtiene un mayor control del estado final del servidor, mientras que las que se han planteado en este apartado ofrecen además otro tipo de posibilidades y servicios.

Hay que mencionar que en un clúster pequeño como en el que se ha trabajado, que solo tiene dos nodos, puede que no se reflejen las ventajas ya que añade cierta complejidad en la parte de gestión a cambio de ofrecer un servicio transparente a los clientes. Sin embargo con un mayor número de máquinas es algo que cobraría mucha importancia ya que posibilita realizar una gestión homogénea del sistema.

Teniendo en cuenta las necesidades planteadas en el apartado *3.1 Requisitos* y el análisis realizado, se ha tomado la decisión de utilizar la herramienta de **Ubuntu MAAS** ya que es la que más se adapta a las necesidades del entorno.

A continuación se plantea en profundidad cuál es la el contexto en el que se ha trabajado y cómo se ha realizado el diseño del sistema, así como sus componentes.

4. Diseño del sistema

Como se ha planteado en el apartado anterior, el objetivo es obtener una **arquitectura tolerante a fallos** que permita el aprovisionamiento de máquinas físicas en base a una serie de requisitos básicos. En esta sección se describe la situación inicial del entorno de trabajo y cómo se ha diseñado para conseguir obtener un clúster con dos máquinas (maestro y esclavo) que funcionen como controladores del servicio de MAAS y de la base de datos PostgreSQL.

Hay que tener presente que se ha trabajado en un **entorno de producción**, en el que en un primer lugar se ha puesto en funcionamiento el servicio de MAAS en una máquina y que en todo momento ha estado operativo y siendo utilizado por usuarios. Ésto supone que la configuración adicional que se ha realizado no debía interrumpir ese servicio y por ello se ha tenido que seguir una metodología diferente a la que se hubiera seguido si se tratara de un entorno de pruebas o de desarrollo. En la *Fig. 4* se muestra un diagrama del estado final del sistema y de cómo están distribuidos los servicios.

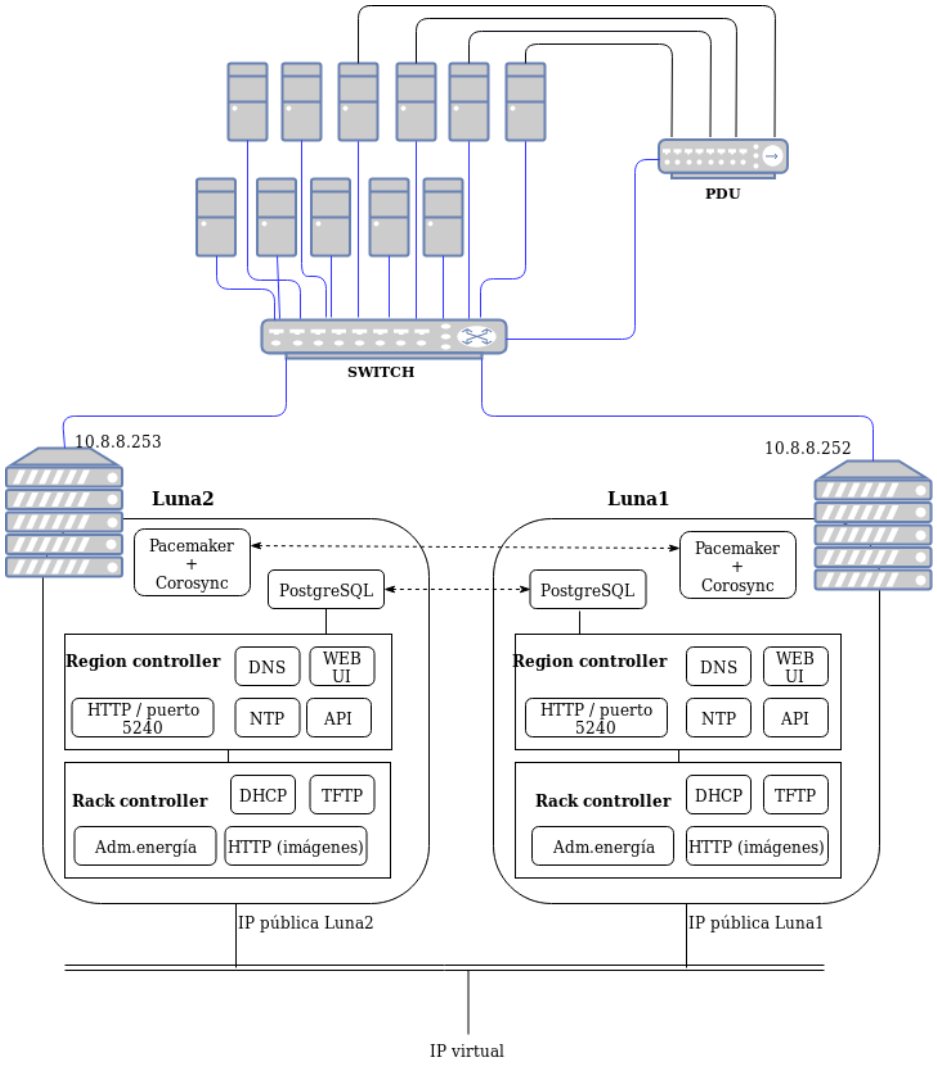

Fig. 4. Diagrama del estado final del sistema.

En el apartado siguiente se especifica la infraestructura del sistema. Los pasos que se han seguido para implementar el sistema se describen en el apartado *4.3 Metodología*, y más en detalle en la sección siguiente *5. Desarrollo*.

4.1 Infraestructura

4.1.1 Hardware

La *Fig. 5* muestra una tabla detallando el hardware que se ha utilizado para implementar la solución. En el caso del maestro se trata de una máquina que ya se encontraba en funcionamiento y sobre la que se ha realizado una instalación previa de la herramienta MAAS.

Hardware	Características
Maestro Controlador del clúster	Sistema operativo Ubuntu v16.04 MAAS v2.3 PostgreSQL v9.5 Dos tarjetas de red
Réplica Réplica del controlador	Inicialmente sin configuración Dos tarjetas de red
Clúster de máquinas	Para gestionar el encendido y el apagado de los equipos es necesario que dichos equipos posean algún tipo de BMC (Board Management Controller: Controlador integrado en la placa base de un ordenador, que permite (entre otras funciones) encender y apagar dicho ordenador de manera remota a través de la red, e independientemente de cualquier sistema operativo), o provean algún otro mecanismo como "wake-on-lan". En este caso todos ellos poseen Intel AMT (un BMC).
Switch	Proporciona la conectividad necesaria entre todas las máquinas.
PDU gestionable	Permite realizar una gestión en remoto del encendido y apagado de ciertas máquinas.

Fig. 5 Hardware utilizado para desarrollar la solución en alta disponibilidad.

4.1.2 Servicios

Además del software que proporciona la capacidad de aprovisionamiento automático es imprescindible integrar otras herramientas que proporcionan al sistema la alta disponibilidad a nivel de la base de datos. El motivo es que la base de datos utilizada por MAAS (PostgreSQL) permite realizar una configuración de replicación, pero deja en las manos del administrador la capacidad de utilizar dicho servicio en alta disponibilidad. Por esta razón ha sido necesario incluir en la configuración dos aplicaciones más: Pacemaker y Corosync.
Finalmente las herramientas software necesarias para gestionar el clúster son las siguientes:

MAAS: se encarga del aprovisionamiento automático de máquinas físicas (encendido, apagado y puesta en marcha) y gestiona los servicios de DNS, DHCP y NTP. Utiliza **PostgreSQL** como base de datos. La arquitectura utilizada por MAAS se muestra en la *Fig. X*.

Dispone de dos componentes principales (*Fig.6*):

- Controlador de región: Se encarga de gestionar las peticiones de los clientes y es el que se encarga de gestionar la base de datos. A su vez ofrece los servicios de DNS (a toda la red) y NTP (a la infraestructura que es controlada por MAAS).
- Controlador de rack: Se encarga de gestionar las máquinas en cuanto a todo lo relacionado con el encendido/apagado, instalación y almacenamiento de las imágenes del sistema operativo. Además proporciona los servicios de DHCP y TFTP.

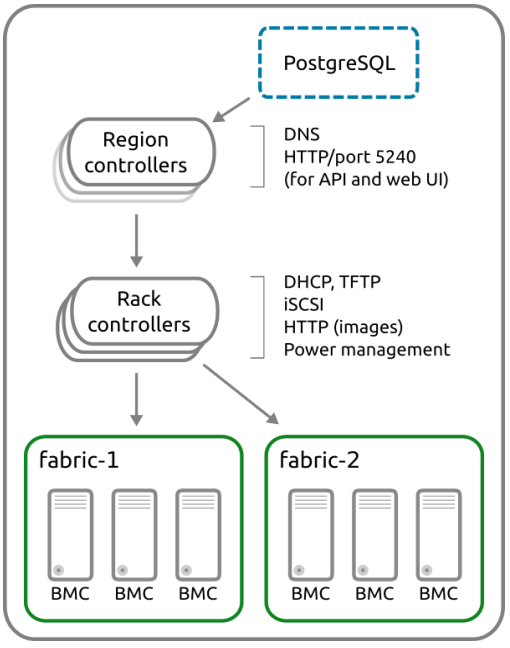

Fig. 6 Arquitectura y servicios de Ubuntu MAAS[15]

[15] Obtenida de la página oficial de Ubuntu MAAS (https://maas.io/).

Tanto estos servicios pertenecientes al paquete de MAAS como la base de datos PostgreSQL pueden ubicarse en máquinas diferentes. Sin embargo en esta ocasión, por motivos de disponibilidad de máquinas físicas para tal efecto y con el fin de simplificar la infraestructura se ha decidido ponerlo en los mismos ordenadores.

Pacemaker: es el responsable de todas las actividades relacionadas con el clúster, como la supervisión de la membresía o la administración de los servicios y los recursos y permite prácticamente cualquier configuración de redundancia. Permite configurar la alta disponibilidad realizando un monitoreo de forma automática, detectando y recuperando los nodos y recursos, y haciendo uso de las capacidades de mensajería proporcionadas por la infraestructura [4].

Corosync: framework utilizado por Pacemaker para gestionar la comunicación entre los nodos del clúster.

PCS: software que proporciona una interfaz de línea de comandos para crear, configurar y controlar cada aspecto de un clúster Pacemaker/Corosync sin tener que entrar a configurar ficheros concretos.

4.2 Red

Como se trata de un cloud privado, se ha decidido que lo más adecuado es que las máquinas que gestiona el clúster dispongan de direcciones IP privadas y que únicamente los controladores posean direcciones en el rango público. Otra configuración posible y que hubiera sido más sencilla, es que todas las máquinas tuvieran una dirección IP pública, pero ésta opción ha sido descartada.

En otro orden de cosas, frecuentemente cuando se diseña un clúster se suelen ubicar ciertos recursos en regiones alejadas geográficamente, cosa que sobre todo es habitual en los clouds públicos (por ejemplo Amazon). En este caso se ha eludido esta parte, aunque sería posible establecer y configurar los servicios (PostgreSQL, MAAS) para que funcionasen en otras zonas.

Hay que tener en cuenta que el clúster que se ha configurado no está pensado para requerir una alta conectividad. Las máquinas que gestiona no necesitan transmitir grandes cantidades de información y las imágenes de los sistemas operativos necesarias para la instalación se descargan únicamente desde los controladores. Además, hay que tener en consideración que el controlador maestro es un punto crítico en la infraestructura de red ya que por una parte se comporta como proxy actuando de pasarela para la red interna de ordenadores, gestiona de forma remota las máquinas y adicionalmente se ocupa de gestionar el switch. En consecuencia se ha valorado que la configuración elegida no va a afectar a las prestaciones del sistema.

Se han diseñado diferentes segmentos de red (*Fig. 7*), permitiendo una mayor legibilidad y organización. En primer lugar se incluyen las subredes 10.8.8.0/24 (dirección de red

principal de los nodos) y 10.9.9.0/24 (dirección de red de administración remota, AMT). Por otra parte se encuentran las direcciones IP públicas correspondientes al maestro y la réplica (y otra que posteriormente funciona como IP virtual alternando entre los dos nodos). Y finalmente la correspondiente a la administración del switch (192.168.0.0/24).

Subred	Funcionalidad
155.210.155.170-171-172 / 24	Subred de IP pública.
10.8.8.0 / 24	Subred de instalación/configuración PCs.
10.9.9.0 / 24	Subred para Intel AMT.
192.168.0.0 / 24	Subred de administración del switch.

Fig. 7 Subredes

4.3 Metodología

Con el objetivo de desarrollar el proyecto de una forma apropiada se ha seguido la metodología propia de la ingeniería (*Fig. 8*). Como se ha mencionado anteriormente se ha realizado un análisis de las herramientas disponibles y un diseño de la infraestructura a utilizar. Posteriormente se lleva a cabo el desarrollo, empezando por preparar y configurar el segundo nodo del clúster (réplica). A continuación y como medida adicional para una posible recuperación se ha realizado una copia de la información y configuración relevante. Más adelante se ha realizado una actualización de los sistemas (sistema operativo, MAAS y PostgreSQL) a las últimas versiones disponibles. Seguidamente se ha implementado la replicación tanto del sistema de MAAS como de la base de datos PostgreSQL, para finalmente configurar la alta disponibilidad de los mismos. Además se ha realizado una verificación individual de cada una de las fases, asegurando el correcto funcionamiento en todo momento. Adicionalmente se ha llevado a cabo una fase de evaluación en la que se comprueba el funcionamiento del sistema, validando la operatividad del mismo.

Hay que recalcar que como se ha mencionado anteriormente, se trata de un sistema en producción, por lo que los pasos a seguir se han adaptado a esta situación, intentando interrumpir el servicio el menor tiempo posible.

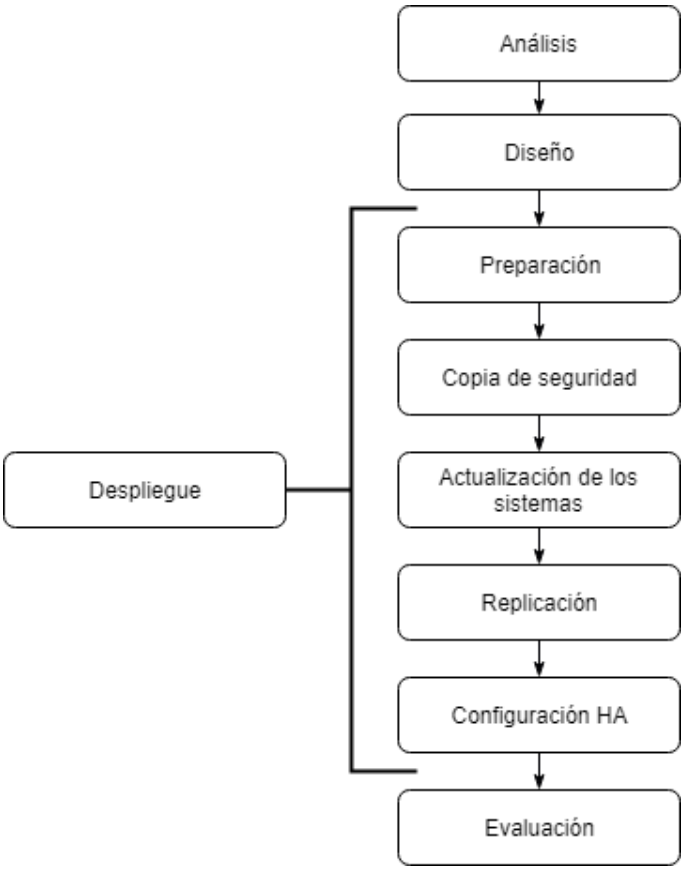

Fig.8 Diagrama con los pasos seguidos en la metodología.

4.3.1 Recuperación

El objetivo de la recuperación es permitir reanudar la operatividad cuando se produce un desastre. En este caso en concreto la recuperación tiene una trascendencia crítica a dos niveles: base de datos y clúster. Por una parte si la base de datos falla en alguno de los nodos conlleva a que la replicación de la información deje de funcionar y con ello el servicio no se ofrece en alta disponibilidad. Por otra parte, si ha habido un fallo en uno de los nodos del clúster se presenta la misma situación.

Dada la importancia de este aspecto, se han incluido en la sección adjunta de Anexos una parte relativa a la a este tema, tanto a nivel de la base de datos (Anexo V) como a nivel de un nodo del clúster que ha fallado (Anexo VI).

5. Desarrollo

Como se ha mencionado, el proyecto se ha realizado completamente trabajando en un **entorno en producción**, poniendo en funcionamiento en primer lugar la herramienta MAAS en un nodo, sobre el sistema operativo Ubuntu 16.04. Para evitar posibles fallos relacionados con las versiones se ha tenido que asegurar que la réplica se configuraba exactamente igual que la máquina principal, para posteriormente realizar una actualización de cada servidor por separado. Finalmente se ha configurado la replicación de los controladores de MAAS y de la base de datos PostgreSQL para que ofrezca alta disponibilidad. Siguiendo esta metodología se ha cerciorado que la funcionalidad operativa del sistema no se viera interrumpida en ningún momento.

En los apartados posteriores se detallan los pasos que se han llevado a cabo en cada una de las máquinas.

5.1 Preparación del maestro

En primera instancia, se ha configurado como una **réplica exacta** de la máquina que estaba funcionando como maestro. La idea que se ha seguido es la de configurar una copia para que en el momento de actualizar el maestro, el sistema siguiera funcionando con la réplica como si no se hubiera modificado nada.

La **instalación del sistema operativo** se ha realizado mediante el instalador proporcionado por Ubuntu en la versión Xenial Xerus (16.04). La siguiente versión de Ubuntu (18.04) estaba disponible pero para evitar problemas de compatibilidad entre versiones se ha instalado la misma que tenía la máquina principal.

5.1.1 Configuración

Como se ha comentado anteriormente, es necesario configurar la réplica con los mismos parámetros exactos que el servidor maestro. Por lo que se ha clonado la configuración a todos los niveles posibles.

- Nombre DNS
- Configuración de las interfaces de red
- Cortafuegos

El fichero de configuración /etc/network/interfaces correspondiente a la réplica se puede encontrar en el Anexo I. La configuración de las interfaces cambia y no se realiza de la forma tradicional. En mismo anexo se detallan los comandos y las modificaciones que se deben realizar tanto para utilizar la nueva nomenclatura como si se quiere seguir utilizando la antigua.

5.1.2 Consideraciones de seguridad

Por motivos de seguridad se ha establecido que el cortafuegos deniegue todas las conexiones entrantes por defecto. Por otra parte se ha permitido el acceso TFTP para todas las conexiones que provengan de las redes 10.8.0.0/16 y 10.9.0.0/16. Además se ha añadido una regla extra para que no se bloqueen los accesos por SSH (para la gestión remota). Los comandos utilizados y las reglas establecidas mediante la herramienta **ufw** se detallan en el Anexo I.

Tras haber realizado toda la configuración anterior, obteniendo una réplica exacta se han instalado los paquetes correspondientes a MAAS y PostgreSQL.

5.2 Preparación de la réplica

Como se ha mencionado ya varias veces, en primer lugar se puso el sistema en funcionamiento en una máquina (maestro). Tal y como se ha descrito en el apartado anterior, se ha configurado otra máquina para que en una primera instancia pueda sustituir a la principal asegurando la continuidad del servicio mientras se realizan las actualizaciones pertinentes. Como medida adicional se ha considerado que era oportuno realizar una copia de los datos de MAAS presentes en el maestro. Estos datos abarcan tanto la configuración como los que se almacenan en la base de datos PostgreSQL. Sin embargo, la herramienta de Ubuntu MAAS no proporciona ninguna herramienta que ayude a realizar esta tarea y la posterior restauración por lo que se ha realizado a mano. Los pasos y comandos necesarios se detallan en el Anexo II.

5.3 Actualización de los sistemas

En una primera instancia se intentó actualizar el nodo principal con las nuevas versiones del sistema operativo, MAAS y PostgreSQL y restaurar sobre éstos la copia de los datos que se había realizado previamente. Sin embargo y tras encontrar múltiples errores en el proceso, se ha determinado que no existe compatibilidad entre las versiones (tanto de MAAS v2.3-v2.4 como PostgreSQL v9.5-v10) por lo que se ha realizado la actualización de manera diferente.

En primer lugar se ha actualizado a la versión de MAAS más alta permitida para Ubuntu 16.04 (v2.3). Una vez se ha comprobado que todo funciona correctamente (adquisición, despliegue, encendido/apagado de máquinas, salida al exterior) se ha actualizado el sistema a la versión de Ubuntu 18.04. Tras realizar este paso se ha procedido a actualizar MAAS a la última versión disponible (2.4), y después de comprobar que los servicios se encuentran en funcionamiento y no hay errores se ha procedido a actualizar la base de datos PostgreSQL (v10).

5.4 MAAS HA

5.4.1 Consideraciones de alta disponibilidad a nivel de rackd

En primer lugar, para que poder gestionar el encendido y el apagado remoto es necesario que los equipos posean algún tipo de **BMC**, o provean algún otro mecanismo como *wake-on-lan*. En este caso todos ellos poseen Intel AMT (un BMC). En el caso de MAAS se identifica automáticamente cuál es el controlador responsable y configura la comunicación.

Por otra parte y como se ha visto previamente, el **DHCP** se ocupa de realizar la **asignación de direcciones IP** a las máquinas gestionadas por el clúster. Dicho servicio ya se había configurado al poner el marcha MAAS en la primera máquina, estableciendo el rango de direcciones IP dinámicas disponibles y reservando dos direcciones IP estáticas correspondientes con los nodos maestro y réplica. En el caso de MAAS, cuando se añade un nuevo controlador se incorpora el software correspondiente permitiendo ofrecer el servicio de DHCP en alta disponibilidad.

5.4.2 Consideraciones de alta disponibilidad a nivel de regiond

Una de las partes más importantes en este nivel es la configuración de la **base de datos** donde MAAS almacena toda la información de estado, y por lo tanto es un punto clave del funcionamiento en alta disponibilidad. Dada su importancia se va a tratar con más detalle en un apartado posterior, pero hay que tener en cuenta que el servicio de PostgreSQL tiene que estar funcionando y replicado para considerar que el sistema ofrece alta disponibilidad.

Por otra parte otro servicio fundamental a nivel de región es el **servidor API** de MAAS. Se debe modificar la configuración de PostgreSQL para permitir al servidor API conectarse a la base de datos de MAAS. Además, para permitir que ambos controladores se comuniquen hay que adaptar la configuración del controlador de región, de manera que sea la misma en ambos servidores.

Finalmente, para asegurar el funcionamiento en alta disponibilidad es necesario configurar una **IP virtual** que va a ser efectiva tanto para MAAS como para el servicio de PostgreSQL. Dicha dirección se va a alternar entre los controladores del clúster según estén actuando como servidor maestro o esclavo y será la que utilicen los clientes para acceder al servicio. Para gestionar cómo se realiza la asignación y el intercambio de la dirección IP entre los servidores se ha utilizado la herramienta de Pacemaker y se especifica más en detalle en el apartado *5.6 Configuración Failover*. Una vez la IP virtual está funcionando, ha sido necesario reconfigurar los servicios de MAAS correspondientes tanto al controlador de rack como al de región para especificar la nueva dirección que se va a usar para la API y la base de datos PostgreSQL. Como en este caso se ha decidido ubicar ambos servicios en las mismas máquinas, solo se ha configurado una dirección IP. En el caso de que se trabajara con la base de datos en otras máquinas habría que gestionar dos direcciones IP diferentes, una para asegurar el funcionamiento en alta disponibilidad del servidor API y otra para la base de datos.

5.5 PostgreSQL

Con el objetivo de conseguir una alta disponibilidad a nivel de los datos previamente se realiza una **replicación** de la base de datos PostgreSQL. Consiste en copiar de forma exacta y mantener actualizados los datos en varios nodos, asegurando que si uno cae, el otro puede seguir respondiendo a las peticiones y manteniendo el sistema activo. Hay que tener en cuenta que la replicación realizada a través de software implica menos costes que la utilización de hardware especializado [9]. El tipo de replicación incluido en el núcleo de PostgreSQL está basado en el **modelo cliente-servidor**, en el que uno de los nodos actúa como maestro y le transmite la información al nodo esclavo, obteniendo así una redundancia de datos. En este caso PostgreSQL dispone de varias herramientas para implementar la replicación. Se pueden clasificar en diferentes modos según:

- Tipo: Basado en triggers (disparadores) o basado en ficheros WAL (Write Ahead Log).
- Técnica: Archivado contínuo (WAL Shipping) o Streaming replication.
- Forma de transmisión: Síncrona o asíncrona.

5.5.1 Modelo de consistencia

Un aspecto fundamental a tener en cuenta en el sistema es la consistencia de los datos. Cuando se genera nueva información o se actualiza en el servidor primario, debe modificarse en el resto de las réplicas. Dado que MAAS trabaja con PostgreSQL hay que estudiar qué posibilidades ofrece y analizar cuál es la más adecuada para este caso. En el sistema que se ha desarrollado, la mayor parte de las operaciones que se producen en la base de datos son de lectura, teniendo en cuenta que además no se producen escrituras concurrentes debido a que los accesos se realizan únicamente al nodo maestro de la base de datos y es éste el que se comunica con el esclavo. Por otro lado, es fundamental que el servicio esté totalmente operativo en caso de que una de las máquinas falle.

En principio, dado que PostgreSQL sólo acepta escrituras en el servidor maestro, la información que contiene siempre va a ser consistente. Sin embargo puede que alguna de las réplicas alcance un estado inconsistente (debido por ejemplo a una partición de red). Según los modelos tradicionales se puede considerar que hay dos tipos de consistencia [1]:

- Consistencia centrada en datos (servidor).
- Consistencia centrada en procesos (clientes).

En el caso que se estudia, PostgreSQL ofrece una consistencia centrada en los datos, y más concretamente se trata de una **consistencia eventual**, en la que todas las réplicas llegan gradualmente a un estado consistente. No obstante puede ofrecer otro tipo de garantías, desde lectura no confirmada a serializable, siendo configuradas mediante parámetros (*default_transaction_isolation*) en el fichero correspondiente.

Por otra parte hay que valorar el tipo de transmisión de los datos. El modelo **síncrono** utiliza unos criterios de corrección estrictos. Los registros de datos se difunden a las réplicas y el servidor maestro espera a que estén escritos antes hacer disponibles dichos datos (mediante un acuse de recibo). Implica un aumento en el tiempo de respuesta debido a la difusión de mensajes, a favor de un compromiso en la consistencia de los datos. En cambio, en el modelo de transferencia **asíncrono** los registros de datos se transmiten del servidor maestro al esclavo sin esperar acuse de recibo y sin utilizar protocolos de difusión. Las réplicas se comunican el con servidor primario y existe un cierto retraso antes de que tengan los datos actualizados. Hay que tener en cuenta que este tipo de transmisión no recomendable para sistemas con un alto número de transacciones y que desee una consistencia secuencial (que todas las réplicas procesen las operaciones en el mismo orden) [10].

Existen diferentes modos de funcionamiento de PostgreSQL con el objetivo de obtener alta disponibilidad. Concretamente para este proyecto se han valorado dos:

> **Failover**: Permite asegurar el funcionamiento de los recursos críticos de manera que en caso de detectarse un fallo, las tareas se derivan automáticamente a una máquina que está en espera (convirtiéndola en principal) y permitiendo así seguir ofreciendo un servicio completamente funcional a los clientes.
>
> **Hot Standby**: Proporciona la capacidad de tener servidores en espera que reciben los datos del servidor primario y son capaces de responder a consultas de solo lectura si el principal falla, haciendo uso de la redundancia de datos.

La configuración *Hot Standby* no se ajusta a lo que se necesita en este caso y sería más interesante si se desea realizar un balanceo de carga. Por este motivo se ha decidido que la opción más adecuada es *Failover*. El tipo de replicación que ha realizado se denomina ***Streaming replication*** y para conseguir la tolerancia a fallos se han utilizado dos herramientas adicionales: Pacemaker y Corosync**.** Este último paso se detalla en el apartado siguiente *5.6. Configuración Failover: Pacemaker + Corosync*.

5.5.2 Replicación en streaming

La replicación en streaming permite transferir registros basados en archivos entre servidor maestro y esclavo, permitiendo que el servidor en espera se mantenga lo más actualizado posible. El principal mecanismo que tiene PostgreSQL para asegurar la durabilidad de los datos es a través éstos **ficheros** denominados **WAL** (Write Ahead Log). Guardan toda la información relativa a transacciones y cambios realizados en la base de datos. Garantizan la integridad y pueden utilizarse para realizar la recuperación frente a posibles inconsistencias en la base de datos tras un fallo del servidor [11]. Estos registros se almacenan en el servidor primario y mediante conexiones TCP/IP se transfieren al nodo réplica. El funcionamiento puede verse reflejado en la *Fig.9*. Otra opción que se ha considerado (aunque ha sido descartada) es habilitar un archivo WAL al que se pudiera acceder en todo momento (en almacenamiento en red), y que conservara los segmentos necesarios.

Para activar el archivo automático de ficheros WAL es necesario definir los parámetros *wal_level*, *archive_mode* y *archive_command* en el fichero de configuración de PostgreSQL. Además, se pueden configurar otras variables como puede ser el tamaño y el número de ficheros que se van a almacenar, todo ello en el servidor maestro. De esta manera el servidor en modo de espera se conecta al primario, que le transmite ficheros WAL a medida que se generan. En el servidor esclavo se debe configurar indicando la información relativa al servidor con el que se tiene que conectar [10]. En el Anexo V se indica paso a paso cómo realizar la instalación y configuración de la base de datos PostgreSQL para obtener y verificar la replicación en modo Streaming.

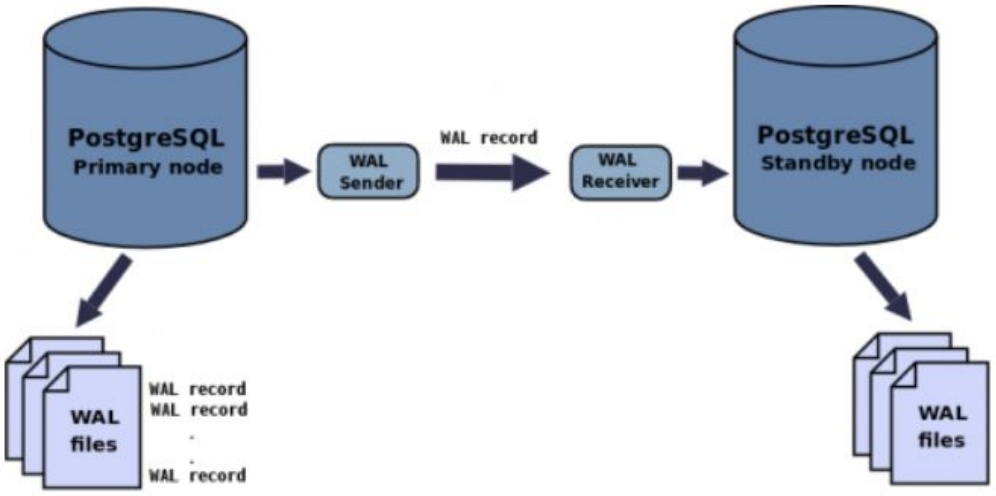

Fig. 9 Streaming replication[16].

Como se ha comentado previamente se pueden configurar dos maneras de transmisión de dichos registros: síncrona y asíncrona. En la forma asíncrona el maestro no espera a que los esclavos hayan recibido los ficheros WAL para realizar el commit. En el modo síncrono el maestro sólo realiza el commit cuando todos los esclavos han recibido correctamente la información.

En el caso que nos ocupa, se ha configurado la **replicación** de manera **asíncrona** por lo que existe un pequeño retraso en la disponibilidad de los datos en el sistema esclavo. Sin embargo el retraso es asumible en éste caso (normalmente está por debajo de un segundo) y el sistema tiene bajos requerimientos (pocas transacciones y cada cierto tiempo). Por otra parte hay que tener en cuenta que la capacidad de almacenamiento de los registros WAL es limitada, por lo que puede ser necesario configurar la variable *wal_keep_segments* en un valor lo suficientemente grande como para garantizar que los segmentos no se reciclen demasiado pronto (antes de que el servidor en espera los haya recibido).

[16] Imagen obtenida de la página https://e-mc2.net/es/hot-standby-y-streaming-replication.

5.6 Configuración Failover: Pacemaker + Corosync

El proceso conocido como **Failover** es aquel en el que si el servidor primario falla, el servidor que está funcionando como réplica debe asumir el papel de maestro para asegurar la disponibilidad del servicio y que el sistema siga funcionando correctamente, sin intervención humana. Lo más común es que ambos sistemas estén conectados mediante algún tipo de mecanismo de latidos que permita verificar la conectividad y viabilidad del servidor primario. En otros casos se suele utilizar un tercer sistema que funciona como testigo, sin embargo esto supone un incremento en la complejidad que no es necesario en todos los escenarios. PostgreSQL no proporciona el software requerido para detectar y notificar un fallo en el servidor maestro por lo que es necesario utilizar una herramienta adicional. En este caso se ha elegido utilizar el mecanismo proporcionado por **Pacemaker + Corosync** ya que cumple perfectamente con los requerimientos y funcionalidades necesarios, teniendo además la ventaja de una comunidad muy activa. La utilización de este software permite obtener una alta disponibilidad ya que posibilita organizar, monitorear y recuperar el servicio en caso de fallo.

5.6.1 Instalación y configuración

Se ha configurado el clúster de forma **activo-pasivo**, de manera que una de las máquinas se encarga de ofrecer el servicio de PostgreSQL y la otra se encuentra en modo de espera y solo pasa a activo cuando la primera falla.

En el primer apartado del Anexo VI se detallan los comandos necesarios para realizar la instalación y configuración de Pacemaker + Corosync.

Para que los servicios funcionen correctamente hay que asegurarse de que se permite el tráfico IGMP y multicast y que los siguientes puertos están abiertos:

Puerto	Servicio
5404, 5405, 5406	Corosync
2224	PCS
3121	Pacemaker

Fig.10 Puertos y servicios de Pacemaker + Corosync

Por razones de administración se recomienda **deshabilitar Pacemaker y Corosync en el arranque del servidor**. Esto ayuda al administrador a la hora de investigar antes de que Pacemaker se inicie y un posible nodo entre en conflicto con otros. Del mismo modo, se deshabilita Corosync para evitar comportamientos inesperados. **Se deberá poner el marcha el servicio de Pacemaker cada vez que caiga un nodo, para que el clúster de PostgreSQL funcione correctamente**.

En el Anexo VI se detallan los comandos necesarios para la creación y configuración del clúster.

Con el fin de manejar el servicio de PostgreSQL en alta disponibilidad se le ha asociado una dirección IP virtual, de manera que va a intercambiarse entre los nodos del clúster, según funcionen como maestro/esclavo. Dicha dirección IP va a ser también utilizada por el servidor API de MAAS de forma que el servidor que actúe como maestro de la base de datos será también el nodo principal de MAAS. Para ello se han creado los recursos: *pgsql, msPostgresql* y *vip-master*.

> ***pgsql***: Define las propiedades de la instancia de Postgresql: dónde se localiza, dónde están sus ficheros de configuración, cómo monitorizarlo, etc.
> ***msPostgresql***: Controla las instancias de Postgresql presentes en el clúster, y decide dónde promociona el primario y dónde el recurso en espera.
> ***vip-master***: Controla la dirección IP virtual asociada al recurso de Postgresql y que va a intercambiarse entre los nodos según funcionen como maestro/esclavo. Inicialmente apunta al nodo que aloja el recurso maestro de Postgresql.
> **fence**: Aísla los recursos cuando se detecta que uno de los nodos no está funcionando bien, permitiendo que el sistema siga funcionando (aunque degradado).

Hay que definir el comportamiento que va a tener la dirección IP virtual para asegurar la alta disponibilidad. El orden de inicio/parada así como el de promover/degradar de los recursos debe de ser asimétrico: se debe mantener la IP maestra en el maestro durante su proceso de degradación para que no se produzcan conflictos en el nodo en espera..

A partir de ahora los servicios que se comuniquen con PostgreSQL (en este caso MAAS) tendrán que hacerlo a través de la dirección IP virtual que ha sido asignada. A su vez, hay que completar la configuración asignando la dirección IP virtual a los servidores de MAAS, tanto a nivel de API como de controlador de rack para asegurar que este servicio también funcione en alta disponibilidad.

Hay que tener en cuenta que tras configurar Pacemaker para proporcionar alta disponibilidad al servicio de PostgreSQL, ambos nodos pueden intercambiarse el papel de maestro/esclavo (en situación de fallo de la máquina principal la otra pasará a ser maestro) por lo que la configuración de los archivos (*pg_hba.conf* y *postgresql.conf*) deberá ser exactamente la misma en las dos máquinas.

5.6.2 Monitorización y fencing

Los dos nodos del clúster están conectados a una red de forma que la comunicación es lo más directa posible y se van monitoreando para ver que todo funciona correctamente. Cuando un nodo detecta que el otro está funcionando mal (uno de los servicios críticos no está en marcha, no responde suficientemente rápido, hay problemas de disco, etc.) tiene que tomar la decisión de convertirse en nodo primario y ocuparse de ejecutar los servicios críticos. Sin embargo, se podría dar el caso en el que ambos servidores detecten que hay

algún problema y decidan a la vez que el otro está funcionando de manera incorrecta, asumiendo el rol de maestro y tomando posesión tanto de la IP como del servicio al mismo tiempo, lo que provocaría un caos en la red. Ésta situación se conoce como *split-brain*. Para evitar esto existe lo que se denomina ***fencing***. El mecanismo que se utiliza en Pacemaker para implementar el *fencing* se llama **STONITH** ("Shoot The Other Node In The Head"), que básicamente significa que un servidor "dispara a la cabeza" de otro nodo, con el objetivo de pararlo y que no se presente el problema que se ha mencionado anteriormente de que ambos asuman el servicio (*split-brain*).

Cuando se está implementado un clúster de alta disponibilidad, el *fencing* es una parte crítica y obligatoria, sin la cual el sistema fallará. Sin embargo, el hecho de parar uno de los nodos implica que el sistema va a funcionar con un clúster degradado y es necesaria una intervención humana para poner en marcha la máquina que ha fallado, solucionar el problema que tenía y volver a poner el clúster en marcha. Una vez se ha hecho esto y de forma automática se vuelve a establecer la configuración de maestro-esclavo tal y como se definió previamente. En el dispositivo que se ha configurado (*external/ssh*) no está disponible la opción de apagar la máquina sino que en vez de eso se reinicia. Además como se ha mencionado anteriormente es necesario poner en marcha el servicio de Pacemaker tras el reinicio de una máquina para que el clúster vuelva a funcionar correctamente. Por ello y para automatizar este proceso se ha realizado un script que permite realizar la recuperación y se incluye en el Anexo VI. La creación y configuración de los mecanismos Stonith se puede encontrar en otro apartado del mismo anexo.

6. Evaluación del sistema

En este apartado se han definido algunos de los posibles escenarios de fallo y cómo actuar en cada caso.

6.1 **Escenario 1.** Intercambio entre maestro/esclavo ante un fallo.

Para comprobar el funcionamiento, se puede apagar el nodo que está actuando como primario. Sin embargo, se puede ejecutar el siguiente comando y ponerlo en espera:

```
$> sudo pcs node standby <nodo>
```

Se puede apreciar que el nodo que anteriormente estaba como esclavo se ha puesto como primario, obteniendo la dirección IP virtual.

```
$> sudo pcs status --full
$> host <IP_virtual>
```

Reanudar la operativa y comprobar el estado.

```
$> sudo pcs node unstandby <nodo>
$> sudo pcs status --full
```

Al consultar el estado se muestra un error informando de que los datos pueden ser inconsistentes (debido a la parada que se ha realizado del recurso de PostgreSQL) y el recurso está parado. Hay que eliminar el fichero de lock que se ha creado en el nodo que se ha puesto en espera y eliminar los errores.

```
$> sudo rm /var/lib/pgsql/tmp/PGSQL.lock
$> sudo pcs resource cleanup msPostgresql
```

Al consultar de nuevo el estado se puede ver que el clúster vuelve a funcionar correctamente con el nuevo nodo maestro.

6.2 **Escenario 2**. Fallo en la base de datos, la base de datos se encuentra en un estado inconsistente.

Para recuperar el nodo que ha sufrido el fallo es necesario en primer lugar detener el servicio de PostgreSQL. A continuación restaurar la base de datos realizando una copia del nodo activo (el que tiene conectividad con el clúster de MAAS). Si el nodo que ha fallado estaba actuando como maestro habrá que eliminar el fichero de lock situado en */var/lib/pgsql/tmp/PGSQL.lock*. Finalmente reiniciar el servicio de PostgreSQL y poner en funcionamiento el servicio de Pacemaker. En el Anexo VI se incluye un script que realiza de forma automática todos estos pasos.

6.3 **Escenario 3**. Fallo en un componente hardware (disco duro, fuente de alimentación). Reemplazo del componente.

Este escenario no ha sido probado estrictamente por motivos evidentes. Sin embargo los pasos que habría que seguir se detallan a continuación.

Si el fallo se trata de un componente hardware como la fuente de alimentación o una tarjeta de red únicamente habría que cambiar el componente. Si en cambio se trata del disco duro habría que reemplazar el componente y aplicar toda la configuración realizada desde el principio (preparación, replicación…). Dado que el clúster seguiría en activo en uno de los nodos no sería necesario crearlo de nuevo (en su defecto se proporciona un script para poner en marcha el clúster completo en el Anexo VI). En cambio sí que sería necesario instalar y configurar el servicio de PostgreSQL aplicando la configuración de maestro tal y como se especifica en el Anexo V. Una vez configurado tal y como se encontraba anteriormente (nombre DNS, configuración de red, etc.) habría que instalar el servicio de Pacemaker, ponerlo en funcionamiento y refrescar la información del clúster para que detecte el nuevo nodo (pcs resource refresh). Si por el contrario el nuevo nodo posee otra configuración (nombre DNS) habrá que darlo de alta en el clúster como si fuera un nuevo recurso.

6.4 **Escenario 4**. Funcionamiento del servicio de MAAS tras el fallo en uno de los nodos del clúster.

El servicio de MAAS está haciendo uso de la dirección IP virtual asignada mediante Pacemaker. Cuando uno de los nodos falla por el motivo que sea, el otro nodo asume dicha IP y sigue ofreciendo el servicio tanto de MAAS como de PostgreSQL. Para comprobarlo se puede poner un nodo en espera como en el escenario 1 y por ejemplo intentar acceder a la interfaz web de MAAS y desplegar una de las máquinas.

6.5 **Escenario 5**. Fallo en mitad de una operación de MAAS.

Se ha verificado el comportamiento del sistema cuando MAAS está en medio de una transacción. Para ello se ha comenzado el despliegue de una máquina y se ha quitado el cable de red conectado al servidor que está funcionando como maestro en ese momento.
Comprobando el estado del clúster se puede apreciar que el nodo aparece “offline” y tras unos segundos se pone el otro nodo como maestro automáticamente, adquiriendo tanto la IP virtual y haciéndose cargo de la base de datos. La máquina que se ha empezado a desplegar queda asignada al usuario que ha empezado a realizar el despliegue y tras unos minutos indica que la instalación ha fallado. Al conectar de nuevo el nodo a la red ambos están como maestros y a continuación, es el nodo que originalmente estaba como principal (el que se ha desconectado de la red) el que se vuelve a estar como primario, pasando el

otro a estar desconectado. Sin embargo no aparece ningún error indicando que la base de datos ha quedado corrupta.

Para recuperarse de esta situación es necesario copiar la base de datos al nodo que aparece como desconectado, además de eliminar el fichero de lock que se ha creado (porque el nodo se estaba ejecutando como maestro cuando se produce el fallo). En el Anexo VI se incluye un script que permite realizar este paso.

El resultado es el mismo si la prueba se realiza en el servidor secundario (exceptuando la parte de eliminación del fichero de lock que al no haber pasado el nodo al rol de maestro no se ha generado).

7. Conclusiones y trabajo futuro

Como se ha ido exponiendo a lo largo de la memoria, en los entornos profesionales de hoy en día se ha convertido en algo fundamental proporcionar la capacidad de soportar fallos y valorar el concepto de alta disponibilidad para los servicios críticos. Sin embargo y como se ha comentado, el coste que supone conseguir esas funcionalidades no es evidente a priori. A su vez hay que señalar la complejidad que conlleva ya que es necesario diseñar el sistema en base a las necesidades específicas de cada entorno. Es necesario enfatizar que en que no es algo trivial, por lo que en primer lugar es imprescindible reconocer los requisitos básicos del sistema. Posteriormente y dado que existen múltiples soluciones disponibles, hay que analizarlas y considerar las implicaciones que conlleva cada decisión que se va a tomar, cosa que a veces tampoco es evidente de antemano.

Por otra parte, hay que valorar que en entornos cloud el aprovisionamiento automático juega un papel fundamental. Permite satisfacer las necesidades de los usuarios suministrando las máquinas bajo demanda y sin el cual, el concepto de cloud computing no tendría sentido.

En otro orden de cosas, hay que mencionar que el hecho de que un servicio ofrezca replicación no significa que su uso en un entorno de alta disponibilidad sea inmediato (como es el caso de PostgreSQL). Como se ha planteado a lo largo del trabajo, hace falta realizar un análisis de las herramientas tecnológicas disponibles y un diseño del sistema para acometer su despliegue. En este caso, se ha necesitado el uso de un servicio de IP virtual, de detección de fallos, entre otros, que, en su conjunto, proporcionen alta disponibilidad. En este caso, se realiza mediante los diferentes recursos suministrados por la herramienta Pacemaker.

Como es lógico, cuando el diseño se ha llevado a la práctica han ido surgiendo los problemas. Sobre todo el factor más influyente en este aspecto es el hecho desarrollar el proyecto en un entorno en fase de producción. Ha sido el principal motivo por el que esta fase ha llevado más tiempo del que en un principio se había estimado. Sin lugar a dudas añade cierta dificultad, obligando a utilizar metodologías que no hubieran sido necesarias si se hubiera desplegado en otro contexto, así como complicaciones adicionales.

Los objetivos de la propuesta se han cumplido en su totalidad, consiguiendo gestionar un conjunto de máquinas físicas de forma automática ofreciendo servicio en alta disponibilidad y garantizando los requisitos de tolerancia a fallos. Sin embargo puede ser interesante ampliar el sistema considerando varias posibilidades:

- Ubicación de la base de datos en otra máquina (actualmente tanto la base de datos como los controladores de aprovisionamiento se encuentran en el mismo servidor físico).
- Adición de máquinas al clúster con el objetivo de establecer el quórum a la hora de tomar decisiones (actualmente sólo son dos nodos).
- Configurar el sistema con HAProxy para realizar balanceo de carga.

Finalmente me gustaría añadir que además de todo lo que he aprendido a todos los niveles, ha sido realmente interesante a nivel personal trabajar tal y como se haría en un entorno profesional a pesar de la dificultad que ha supuesto. Adicionalmente, acostumbrada a trabajar en equipo, el hecho de desarrollar el proyecto completamente de forma individual y abarcar todos los aspectos ha sido un reto.

8. Bibliografía

[1] *Andrew S. Tanenbaum y Maarten Van Steen (2007).* Distributed systems. Principles and paradigms.
[2] *Valter Balegas; Cheng Li; Mahsa Najafzadeh; Daniel Porto; Allen Clement; Srgio Duarte; Carla Ferreira; Johannes Gehrke; João Leitão; Nuno Preguia; Rodrigo Rodrigues; Marc Shapiro y Viktor Vafeiadis (2016).* Geo-replication: Fast If Possible, Consistent If Necessary.
[3] *Patricia T. Endo; Moisés Rodrigues; Glauco E. Gonçalves; Judith Kelner; Djamel H. Sadok y Calin Curescu.* High availability in clouds: systematic review and research challenges
[4] *Luka Perkov; Nikola Pavković y Juraj Petrović (2011).* High-Availability Using Open Source Software
[5] *Hennessey, Jason; Hill, Chris; Denhardt, Ian; Viggnesh, Venugopal; Silvis, George; Krieger, Orran y Desnoyers, Peter (2014).* Hardware as a service - enabling dynamic, user-level bare metal provisioning of pools of data center resources.
[6] Peter Mell y Timothy Grance (2011). The NIST definition of Cloud Computing.
[7] *P. Rad, A T Chronopoulos, P. Lama, P. Madduri, C. Loader (2015).* Benchmarking Bare Metal Cloud Servers for HPC Applications
[8] *Charalampos Gavriil Kominos; Nicolas Seyvet y Konstantinos Vandikas (2017).* Bare-metal, virtual machines and containers in OpenStack.
[9] *R. Guerraoui y A. Schiper (1997).* Software-based replication for fault tolerance.
[10]*Hans-Jürgen Schönig (2015).* PostgreSQL Replication.
[11] *Shaun M. Thomas (2014).* PostgreSQL 9 High Availability Cookbook.

Enlaces de interés

Redundancia y alta disponibilidad
http://blogs.salleurl.edu/networking-and-internet-technologies/alta-redundancia-y-disponibilidad-i/

Bare metal provisioning tools
https://www.zenlayer.com/bare-metal-provisioning-tools/
https://devops.com/flap-part-1-server-provisioning/

MAAS High Availability
https://docs.maas.io/2.1/en/manage-ha

PostgreSQL High Availability
https://www.postgresql.org/docs/9.5/static/high-availability.html

Replicación
https://e-mc2.net/es/hot-standby-y-streaming-replication
https://blog.powerdata.es/el-valor-de-la-gestion-de-datos/beneficios-de-la-replicacion-de-base-de-datos

Gestión del clúster mediante Pacemaker
https://pgstef.github.io/2018/02/07/introduction_to_postgresql_automatic_failover.html